Floating-point round-off error analysis of safety-critical avionics software

Laura Titolo

National Institute of Aerospace

CSV 2022

- Writing correct FP code is challenging!
- Round-off errors $\Rightarrow$ computed $\mathbb{F}$ result $\neq$ expected $\mathbb{R}$ result
- Unstable guards $=$ $\mathbb{F}$ control-flow $\neq$ $\mathbb{R}$ control-flow
  $\Rightarrow$ Large divergence between real and FP results
- Round-off errors in safety-critical avionics software:
  - ADS-B Compact Position Reporting (CPR)
    $\Rightarrow$ wrong position determination
  - Air traffic detect-and-avoid systems
    $\Rightarrow$ resolution maneuvers that are not implicitly coordinated
  - Geofencing in autonomous UAS
    $\Rightarrow$ incorrect determination of being inside/outside a geofence (point-in-polygon)

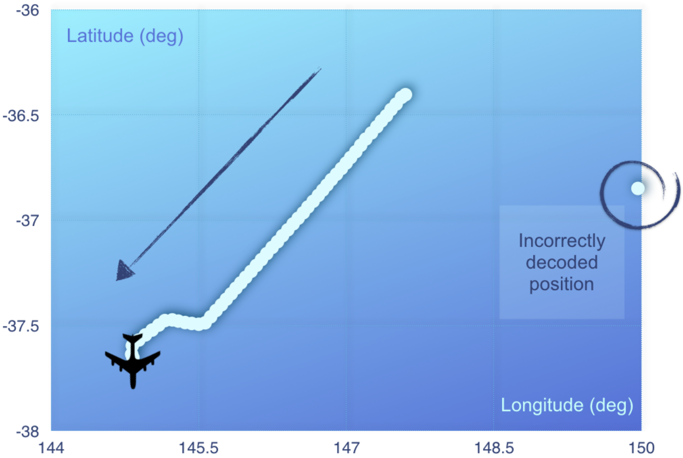
Floating-Point Round-off Errors

- Writing correct FP code is challenging!
- Round-off errors $\Rightarrow$ computed $\mathbb{F}$ result $\neq$ expected $\mathbb{R}$ result
- Unstable guards = $\mathbb{F}$ control-flow $\neq$ $\mathbb{R}$ control-flow
  $\Rightarrow$ Large divergence between real and FP results
- Round-off errors in safety-critical avionics software:
  - ADS-B Compact Position Reporting (CPR)
    $\Rightarrow$ wrong position determination
  - Air traffic detect-and-avoid systems
    $\Rightarrow$ resolution maneuvers that are not implicitly coordinated
  - Geofencing in autonomous UAS
    $\Rightarrow$ incorrect determination of being inside/outside a geofence (point-in-polygon)

ADS-B

- Automatic Dependent Surveillance - Broadcast (ADS-B)
- Supports the Next generation of air traffic management systems (NextGen)
- Aircraft periodically broadcasts accurate surveillance information to ground stations and near aircraft position and velocity
- Automatic (no pilot intervention needed) and dependent on navigation system
- Mandatory from Jan 1, 2020 (in USA and Europe)
- More than 40000 aircraft currently equipped

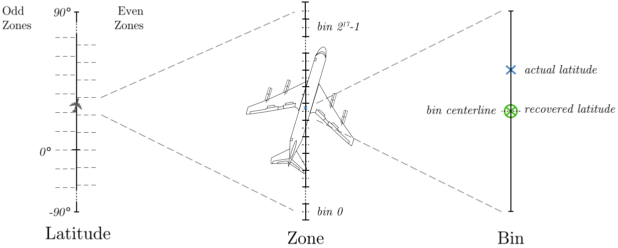
- CPR (Compact Position Reporting) is responsible for decoding and encoding the position of the aircraft in ADS-B
- CPR encodes the aircraft position in 35 bits such that, for airborne applications, the decoded position is intended to guarantee a position accuracy of approximately 5 m
- Problem: pilots and manufacturers have reported errors in the positions obtained by encoding and decoding with the CPR algorithm

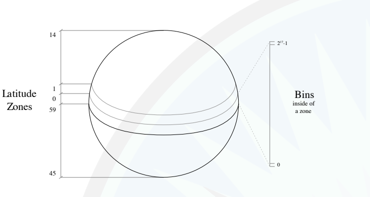


Reported by Airservices Australia (2007)

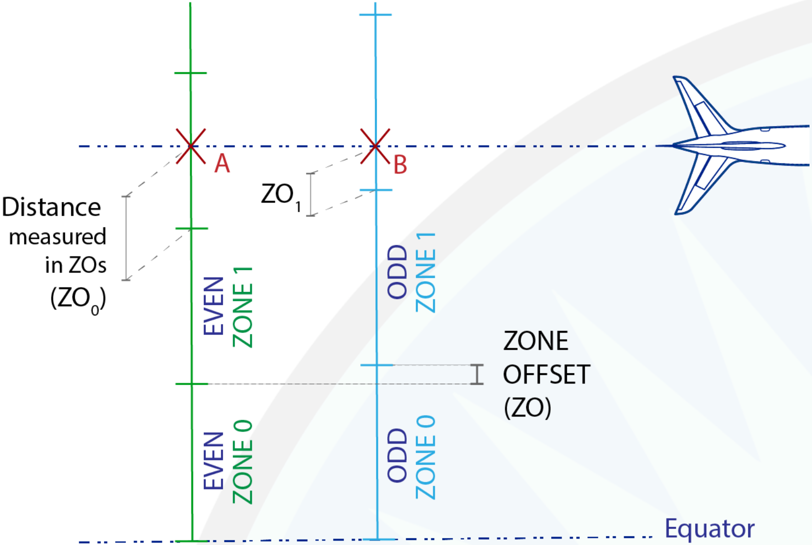
To encode lat, calculate:

1. Distance from southern edge of enclosing zone
   - $\text{mod}\,(\mathrm{lat}, \mathrm{Dlat})$
2. Proportion w.r.t. the entire zone
   - $\text{mod}\,(\mathrm{lat}, \mathrm{Dlat}) \cdot \frac{1}{\mathrm{Dlat}}$
3. Correspondent *bin* number
   - $\text{mod}\,(\mathrm{lat}, \mathrm{Dlat}) \cdot \frac{1}{\mathrm{Dlat}} \cdot 2^{17}$
4. Round to the nearest integer
   - $\mathrm{ZY} = \left\lfloor \text{mod}\,(\mathrm{lat}, \mathrm{Dlat}) \cdot \frac{1}{\mathrm{Dlat}} \cdot 2^{17} + \frac{1}{2} \right\rfloor$


Odd Zones
90°
Even Zones
0°
-90°
Latitude
bin 2^{17}-1
bin 0
Zone
actual latitude
bin centerline
recovered latitude
Bin

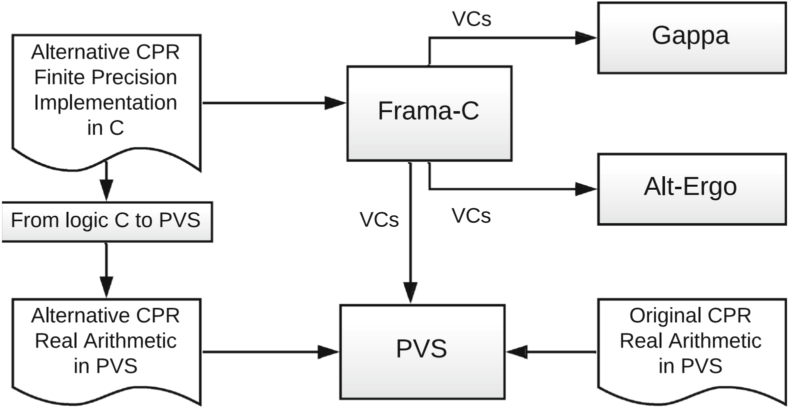

Latitude Zones
14
1
0
59
45
2^{17}-1
0
Bins
inside of
a zone

CPR Decoding



Our solution

- Use a suite of formal methods tools to provide a verified and correct implementation of CPR with finite precision arithmetic:
  - PVS and Coq: interactive theorem provers
  - Gappa: framework for the analysis of floating-point programs
  - Frama-C: static analysis suite for C



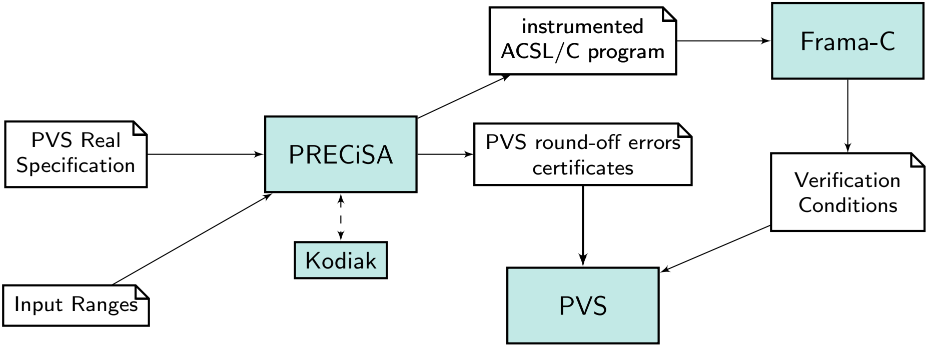
- Logic ACSL declarations translated to PVS by hand proved equivalent to existent CPR formalization
- C code verified using Frama-C/WP/Alt-Ergo/Gappa

CPR verification

- Problem: Counterexamples found for both decoding settings Even Assuming (exact) real-valued arithmetic
- Solution: New more restrictive requirements proposed
- Problem: Use of several numerically unstable operators (floor and module)
- Solution: Proposed simpler formulation reducing numerical complexity
- Prototype implementation formally verified C, PVS, Frama-C, Gappa, Alt-Ergo
- The verified implementation is included in the revised version of the ADS-B standards document as the reference implementation of the CPR algorithm (RTCA DO-260B/Eurocae ED-102A)

Making the approach more automatic

- The CPR verification was not completely automatic!
- Idea: Automatically generate and verify a floating-point C implementation from a PVS real numbers specification which is
  - instrumented to detect unstable guards
  - annotated with information about round-off errors that may occur
- Integrate three formal methods tools
  - PRECiSA: framework for the analysis of floating-point programs
  - PVS: interactive theorem prover
  - Frama-C: static analysis suite for C


instrumented ACSL/C program
Frama-C
PVS Real Specification
PRECiSA
PVS round-off errors certificates
Verification Conditions
Kodiak
Input Ranges
PVS

PVS Real Specification

$$\boxed{\begin{aligned} P = \mathit{if}\ & x * y \geq z \\ & \mathit{then}\ 1 \\ & \mathit{else}\ -1 \end{aligned}}$$

PVS Real Specification

$$P = \mathit{if}\ \ x * y \geq z \ \ \mathit{then}\ 1 \ \ \mathit{else}\ -1$$

$\downarrow$ Real to Floating-Point

Floating-Point Implementation

$$\widetilde{P} = \mathit{if}\ \ \tilde{x}\tilde{*}\tilde{y} \geq \tilde{z} \ \ \mathit{then}\ 1 \ \ \mathit{else}\ -1$$

- $\tilde{x}\tilde{*}\tilde{y} \geq \tilde{z}$ may evaluate differently from $x * y \geq z$ due to round-off errors
- the divergence is $|P - \widetilde{P}| \leq |1 - (-1)| = 2$

$$\begin{array}{c}\textbf{PVS Real}\ \text{Specification}\\ \Big\downarrow \text{Real to Floating-Point}\\ \textbf{Floating-Point}\ \text{Implementation}\\ \Big\downarrow \tau\ \text{instrumentation}\\ \widetilde{P}^{\tau} = \textbf{Instrumented}\ \text{version of } P \text{ detecting unstable guards}\end{array}$$

- τ replaces the guards in the conditionals with more restrictive ones

$$\begin{array}{l} \textbf{if } \tilde{x} \tilde{*} \tilde{y} \geq \tilde{z} \\ \quad \textbf{then } 1 \\ \quad \textbf{else } -1 \end{array} \xrightarrow{\quad \tau \quad} \begin{array}{l} \textbf{if } \tilde{x} \tilde{*} \tilde{y} \tilde{-} \tilde{z} \geq \epsilon \\ \quad \textbf{then } 1 \\ \textbf{elsif } \tilde{x} \tilde{*} \tilde{y} \tilde{-} \tilde{z} < -\epsilon \\ \quad \textbf{then } -1 \\ \quad \textbf{else } \omega \end{array}$$

- If $\tau(P)$ does not return a *warning* ω
  - $\Rightarrow$ P returns the same value
  - $\Rightarrow$ P's execution is stable
- If P's execution is unstable
  - $\Rightarrow$ $\tau(P)$ returns a *warning* ω
- Over-approximation $\Rightarrow$ false alarms

- τ replaces the guards in the conditionals with more restrictive ones

$$\begin{array}{l}\textbf{if } \tilde{x}\tilde{*}\tilde{y} \geq \tilde{z} \\ \quad \textbf{then } 1 \\ \quad \textbf{else } -1\end{array} \xrightarrow{\quad\tau\quad} \begin{array}{l}\textbf{if } \tilde{x}\tilde{*}\tilde{y}\tilde{-}\tilde{z} \geq \epsilon \\ \quad \textbf{then } 1 \\ \textbf{elsif } \tilde{x}\tilde{*}\tilde{y}\tilde{-}\tilde{z} < -\epsilon \\ \quad \textbf{then } -1 \\ \quad \textbf{else } \omega\end{array}$$

over-approx
round-off
error of $\tilde{x}\tilde{*}\tilde{y}\tilde{-}\tilde{z}$

- If $\tau(P)$ does not return a *warning* ω
  - $\Rightarrow$ P returns the same value
  - $\Rightarrow$ P's execution is stable
- If P's execution is unstable
  - $\Rightarrow$ $\tau(P)$ returns a *warning* ω
- Over-approximation $\Rightarrow$ false alarms

- τ replaces the guards in the conditionals with more restrictive ones

$$\begin{array}{l}\textbf{if } \tilde{x}\tilde{*}\tilde{y} \geq \tilde{z}\\ \quad\textbf{then } 1\\ \quad\textbf{else } -1\end{array} \xrightarrow{\quad\tau\quad} \begin{array}{l}\textbf{if } \tilde{x}\tilde{*}\tilde{y}\tilde{-}\tilde{z} \geq \epsilon\\ \quad\textbf{then } 1\\ \textbf{elsif } \tilde{x}\tilde{*}\tilde{y}\tilde{-}\tilde{z} < -\epsilon\\ \quad\textbf{then } -1\\ \quad\textbf{else } \omega\end{array}$$

- If $\tau(P)$ does not return a *warning* ω
  - $\Rightarrow$ P returns the same value
  - $\Rightarrow$ P's execution is stable
- If P's execution is unstable
  - $\Rightarrow$ $\tau(P)$ returns a *warning* ω
- Over-approximation $\Rightarrow$ false alarms

P = **PVS Real** specification

$$\big\downarrow \text{ Real to Floating-Point}$$

$\widetilde{P}$ = **Floating-Point** implementation of P

$$\big\downarrow \ \tau \text{ instrumentation}$$

$\widetilde{P}^{\tau}$ = **Instrumented** version of P detecting unstable guards

P = **PVS Real** specification

$\downarrow$ Real to Floating-Point

$\widetilde{P}$ = **Floating-Point** implementation of P

$\downarrow$ τ instrumentation

$\widetilde{P}^{\tau}$ = **Instrumented** version of P detecting unstable guards

$\downarrow$ C code generation + round-off error estimation

C implementation of $\widetilde{P}^{\tau}$ with **ACSL annotations** on the round-off error

Example: tcoa

- $tcoa$ is used in the library DAIDALUS (Detect-and-avoid) to compute the time to co-altitude of two aircraft

Time to co-altitude

$$tcoa(s_z, v_z) = if\ s_z v_z < 0\ then\ -(s_z/v_z)\ else\ 0$$

Transformed Time to co-altitude

$$\begin{aligned}\widetilde{tcoa}^{\tau}(\tilde{s}_z, \tilde{v}_z, e_{tcoa}) = {} & if\ \tilde{s}_z\tilde{v}_z < -e_{tcoa}\ then\ -(\tilde{s}_z \widetilde{/} \tilde{v}_z) \qquad \%|(\tilde{s}_z\tilde{v}_z) - (s_z v_z)| \leq e_{tcoa} \\ & elsif\ \tilde{s}_z\tilde{v}_z \geq e_{tcoa}\ then\ 0\ \ else\ \omega\end{aligned}$$

C code generation: symbolic function

$$
\begin{aligned}
&/{*}@\ real\ \ tcoa(real\ \ s_z, real\ \ v_z) = s_z * v_z < 0\ ?\ -(s_z/v_z) : 0\\
&\quad double\ \ fp\_tcoa(double\ \ \tilde{s}_z, double\ \ \tilde{v}_z) = \tilde{s}_z \tilde{*} \tilde{v}_z < 0\ ?\ \tilde{-}(\tilde{s}_z \tilde{/} \tilde{v}_z) : 0\\
&\quad predicate\ \ tcoa\_stable\_paths(real\ \ s_z, real v_z, double\ \ \tilde{s}_z, double\ \ \tilde{v}_z) =\\
&\qquad (v_z \neq 0 \wedge s_z * v_z < 0 \wedge \tilde{v}_z \neq 0 \wedge \tilde{s}_z \tilde{*} \tilde{v}_z < 0) \vee (s_z * v_z \geq 0 \wedge \tilde{s}_z \tilde{*} \tilde{v}_z \geq 0)\\
&\quad \textbf{requires} : 0 \leq e\\
&\quad \textbf{ensures} : result \neq \omega \Rightarrow (result = fp\_tcoa(\tilde{s}_z, \tilde{v}_z)\\
&\quad \wedge \forall s_z, v_z (|(\tilde{s}_z \tilde{*} \tilde{v}_z) - (s_z * v_z)| \leq e \Rightarrow tcoa\_stable\_paths(s_z, v_z, \tilde{s}_z, \tilde{v}_z))\\
&\ {*}/
\end{aligned}
$$

$$
\begin{aligned}
&double\ \ tau\_tcoa\ (double\ \ \tilde{s}_z, double\ \ \tilde{v}_z, double\ \ e)\{\\
&\quad if\ \ (\tilde{s}_z \tilde{*} \tilde{v}_z < -e)\{\\
&\quad return\ \tilde{-}(\tilde{s}_z \tilde{/} \tilde{v}_z);\\
&\quad \}\ else\ \{\ if\ \ (\tilde{s}_z \tilde{*} \tilde{v}_z \geq e)\\
&\qquad\quad \{return\ \ 0;\\
&\qquad\quad \}\ else\ \{return\ \ \omega;\}\}\}
\end{aligned}
$$

transformed program

$/{*}@\ real\ \ tcoa(real\ \ s_z, real\ \ v_z) = s_z * v_z < 0\,?\, -(s_z/v_z) : 0$

$double\ \ fp\_tcoa(double\ \ \tilde{s}_z, double\ \ \tilde{v}_z) = \tilde{s}_z \tilde{*} \tilde{v}_z < 0\,?\, \tilde{-}(\tilde{s}_z \tilde{/} \tilde{v}_z) : 0$

$predicate\ \ tcoa\_stable\_paths(real\ \ s_z, real v_z, double\ \ \tilde{s}_z, double\ \ \tilde{v}_z) =$

$(v_z \neq 0 \wedge s_z * v_z < 0 \wedge \tilde{v}_z \neq 0 \wedge \tilde{s}_z \tilde{*} \tilde{v}_z < 0) \vee (s_z * v_z \geq 0 \wedge \tilde{s}_z \tilde{*} \tilde{v}_z \geq 0)$

requires : $0 \leq e$

ensures :$result \neq \omega \Rightarrow (result = fp\_tcoa(\tilde{s}_z, \tilde{v}_z)$

$\wedge \forall s_z, v_z (|(\tilde{s}_z \tilde{*} \tilde{v}_z) - (s_z * v_z)| \leq e \Rightarrow tcoa\_stable\_paths(s_z, v_z, \tilde{s}_z, \tilde{v}_z))$

$*/$

$double\ \ tau\_tcoa\ \ (double\ \ \tilde{s}_z, double\ \ \tilde{v}_z, double\ \ e)\{$

$if\ \ (\tilde{s}_z \tilde{*} \tilde{v}_z < -e)\{$

$return\ \ \tilde{-}(\tilde{s}_z \tilde{/} \tilde{v}_z);$

$\}\ else\ \{\ if\ \ (\tilde{s}_z \tilde{*} \tilde{v}_z \geq e)$

$\{return\ \ \ 0;$

$\}\ else\ \{return\ \ \ \omega;\}\}\}$

post-condition

- symbolic functions do not depend on initial ranges for the input vars
- PRECiSA uses the global optimizer Kodiak to maximize the symbolic error expression given these ranges

initial values arguments

$$
\begin{aligned}
/*@\textbf{ensures}: &\forall s_z, v_z (1 \leq s_z \leq 1000 \wedge 1 \leq v_z \leq 1000 \wedge \\
& |\tilde{s}_z - s_z| \leq ulp(s_z)/2 \wedge |\tilde{v}_z - v_z| \leq ulp(v_z)/2) \wedge \\
& result \neq \omega \\
& \Rightarrow |result - tcoa(s_z, v_z)| \leq 2.78e-12 \\
& */
\end{aligned}
$$

$$
\begin{aligned}
&double\ tau\_tcoa\_num(double\ \tilde{s}_z, double\ \tilde{v}_z)\{ \\
&\quad return\ \ tau\_tcoa\ (\tilde{s}_z, \tilde{v}_z, 1.72e-10)\}
\end{aligned}
$$

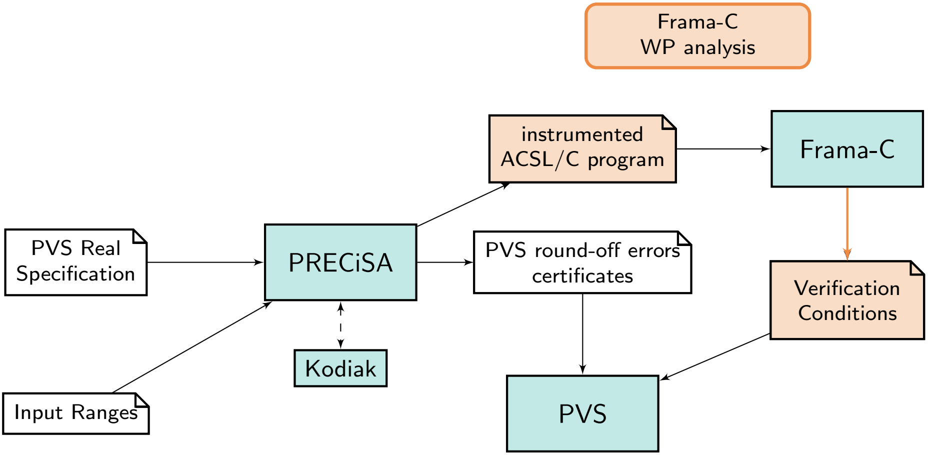
- symbolic functions do not depend on initial ranges for the input vars
- PRECiSA uses the global optimizer Kodiak to maximize the symbolic error expression given these ranges

$$
\begin{aligned}
/*@\textbf{ensures}:& \forall s_z, v_z (1 \le s_z \le 1000 \wedge 1 \le v_z \le 1000 \wedge \\
& |\tilde{s}_z - s_z| \le ulp(s_z)/2 \wedge |\tilde{v}_z - v_z| \le ulp(v_z)/2) \wedge \\
& result \neq \omega \\
& \Rightarrow |result - tcoa(s_z, v_z)| \le 2.78e-12 \\
& */
\end{aligned}
$$

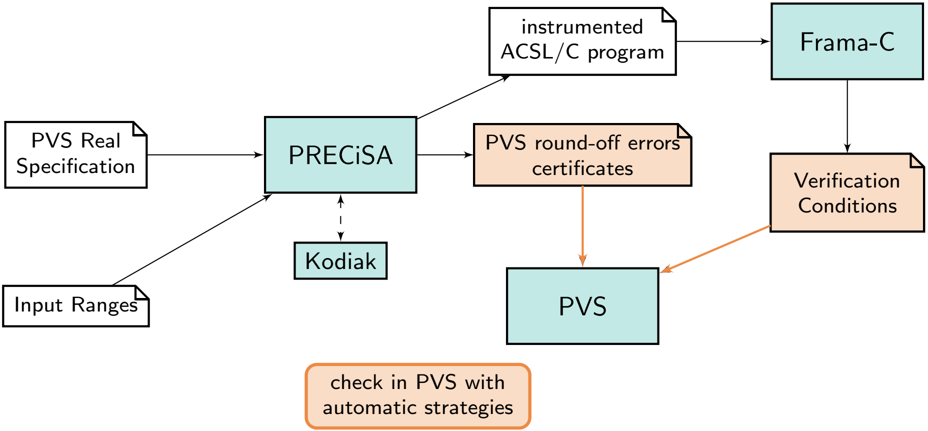
$$
\begin{aligned}
& double\ tau\_tcoa\_num(double\ \tilde{s}_z, double\ \tilde{v}_z)\{ \\
& \quad return\ \ tau\_tcoa\ (\tilde{s}_z, \tilde{v}_z, 1.72e-10)\}
\end{aligned}
$$

round-off error
of tau_tcoa_num
computed by PRECiSA

C code verification



C code verification



Conclusions

- Successful integration of formal methods tools
- Generation of C code from a PVS real specification instrumented to detect unstable guards
- Automatic verification with Frama-C+PVS
  $\Rightarrow$ no user expertise required in FP arithmetic or theorem proving
- PRECiSA is available under NASA Open Source Agreement (`http://github.com/nasa/precisa`)
- Application to significant fragments of the NASA formalizations of PolyCARP (geofencing) and DAIDALUS (detect-and-avoid)

References

- PVS (https://coq.inria.fr/)
- Coq (https://pvs.csl.sri.com/)
- Gappa (https://gappa.gitlabpages.inria.fr/)
- Frama-C (https://frama-c.com/)
- Alt-Ergo (https://alt-ergo.ocamlpro.com/)
- Kodiak (https://github.com/nasa/Kodiak)
- DAIDALUS (https://github.com/nasa/daidalus)
- PolyCARP (https://software.nasa.gov/software/LAR-18798-1)

Thanks for your attention!
Questions?